2152

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re the application of:

Mark LYNCH et al

Serial No.: 09/920943 Group Art Unit: 2152

Filed: August 3, 2001

For: E-BUSINESS MOBILITY PLATFORM

CLAIM OF PRIORITY
UNDER 35 U.S.C. § 119

Commissioner of Patents
Washington, D.C. 20231

Sir:

The benefit of the filing date of prior foreign application Nos. 01650013.4; 00650094.6; and 00650206.6 all filed in Europe on February 6, 2001; August 4, 2000; and December 11, 2000, respectively, is hereby requested and the right of priority provided in 35 U.S.C. §119 is hereby claimed.

In support of this claim, filed herewith is a certified copy of said original foreign applications.

Respectfully submitted,

JACOBSON HOLMAN, PLLC

By: _____
John C. Holman
Reg. No. 22,769

50

400 Seventh Street, N.W.
Washington, D.C. 20004-2201
Telephone: (202) 638-6666
Atty. Docket No.: P67024US0
Date: November 2, 2001

Europäisches Patentamt	European Patent Office	Office européen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

01650013.4

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 26/07/01
LA HAYE, LE

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 01650013.4
Demande n°:

Anmeldetag:
Date of filing: 06/02/01
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):

Mobileaware Technologies Limited
Citywest Campus, Dublin 24
IRELAND

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:

A content transformation system and method

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:

/

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE/TR
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

"A content transformation system and method"

Introduction

The invention relates to delivery of content from servers to end-user devices of a variety of different types such as those using WML, HTML, SMS, and E-mail communication capabilities.

In its original form, the Internet succeeded in bringing communities of users together with services on a truly global scale. However, interaction with the Internet was largely restricted to individuals with access to a PC and HTML browser.

The next phase of evolution of the Internet enables mobile phone users to interact with services over WAP (Wireless Application Protocol). Additionally we are witnessing the emergence of a wide variety of devices – including Personal Digital Assistants (PDAs), Digital Television, Games Consoles and Voice Browsers – which provide access to the Internet.

In the first phase of the Internet, services were targeted singularly at HTML browsers – HTML authors created pages of content for the sole purpose of being rendered on a PC browser. As WAP gains popularity, service providers are required to support two versions of each service – one for WAP; one for HTML. As each of the devices outlined above gains popularity, service providers will find themselves facing the exponentially greater challenge of managing a wealth of device-specific versions of their services.

In order to achieve communication with a range of different user devices it is desirable to perform transcoding from the markup language of the content provider to that of the devices. This would avoid the need to author content more than once, and would also allow content authors to continue to work in the languages with

which they are familiar despite changes in and additions to device languages. It would also avoid the complexity and overhead involved in authoring each page several times.

There is at present a technology for transcoding, namely X SLT (XML Stylesheet Transformation). However this does not cater well for transcoding to user devices because it has limited restructuring capabilities.

The invention is directed towards providing a method and system for effective, versatile, and real-time transcoding so that there is only a requirement for content to be authored once.

Statements of Invention

According to the invention, there is provided a transcoder comprising:-

a request handler for receiving a request from a user device for content from a server, and for retrieving the requested content, and

a transformation means for dynamically transcoding the content to a format compatible with the device.

In one embodiment, the transformation means comprises means for transcoding from a task markup language format describing the content inherent in a number of formats such as HTML, XHTML basic, WML, and plain text.

In another embodiment, the request handler comprises means for generating a call chain of request-handling components chosen according to the type of user device.

In a further embodiment, the transformation means comprises means for pooling call chains for re-use.

In one embodiment, the transformation means comprises means for using both user personalisation data and device data to perform transcoding.

In another embodiment, the transcoder comprises a plurality of accessors, each comprising means for retrieving data in a manner which isolates the call chain components from the device data and personalisation data databases.

In one embodiment, a thin accessor is instantiated as a lightweight façade to an object-orientated class.

In another embodiment, an accessor is not instantiated unless it is required, "lazy-evaluated".

In a further embodiment, the transformation means comprises means for pooling instantiated accessors.

In one embodiment, the transformation means comprises means for caching transformed content, and for preventing transformation from taking place if relevant content is in a cache.

In another embodiment, the transformation means comprises means for checking a cache at a plurality of stages in a call chain.

In a further embodiment, the transformation means comprises means for building a Document Object Model (DOM) tree.

In one embodiment, the tree is built with elements which have transformation intelligence.

According to another aspect, the invention provides a method of providing content to a user device, the method comprising the steps of:-

authoring content in a generic markup language; and

transforming the content on-the-fly in real time to a device format according to user preferences and device capabilities.

In one embodiment, the content is aggregated with content in a specific markup language before transformation.

Detailed Description of the Invention

The invention will be more clearly understood from the following description of some embodiments thereof, given by way of example only with reference to the accompanying drawings in which:-

Fig. 1 is a diagram illustrating signal stages between content hosts and end user devices;

Fig. 2 is a diagram showing the stages in more detail;

Fig. 3 is a diagram of transcoder;

Fig. 4 is a diagram illustrating a prior art approach to handling a request;

Fig. 5 is a diagram illustrating handling of request according to the invention;

Fig. 6 is a diagram illustrating operation of accessors of the transcoder;

Fig. 7 is a diagram illustrating content cacheing in the transcoder;

Fig. 8 is a high-level diagram illustrating a request/response sequence using a Document Object Model (DOM) of the prior art; and

Fig. 9 is a diagram illustrating DOM tree structures.

Referring to Fig. 1 communication between the user device domain and the content host domain comprises content "get", content merging and aggregation; contentent transformation; content personalisation, and delivery stages. The hosting stage involves hosting a single canonical representation in a Task Markup Language (TML).

Referring to Fig. 2, in the host-to-device stages the following are implemented:

- a Web authoring tool creates a TML page, which is XHTML plus TML mark-up tags,
- the content is stored on the server in TML format,
- create non-TML content which is routed through content accessors,
- all content (TML and non-TML) is aggregated by a content aggregator, and

- an optimised transcoder receives the aggregated content and uses stored end-user mobility preferences and device capabilities to provide the end-user device formats such as WML, HTML, and CHTML.

The TML tags added at the authoring stage for TML content capture additional presentational information in a manner decided by the author. The tagging depends on how the information is to be presented to the end user. For example, one type of tagging is used where there is dynamic content flowing into a template page such as news feeds. In this example the author constructs a template exactly as would be done for a HTML-only site and then, using the same tool, incorporates additional markup to control the delivery of information. These tags operate in real time to convert (say, XML) to TML and incorporate it into the template for delivery to the client.

In more detail, TML (Task Markup Language) is a generic non-device specific language used to describe the content inherent in a number of content presentation formats including HTML, XHTML basic, WML, plain text, and heterogeneous business objects. It is a superset of XHTML. TML adds tags to XHTML to mark-up the structure of XHTML documents. The addition of these tags enables a non-device specific description of content. TML is XML compliant.

TML allows content pages to be easily divided into parts, which can then be extracted as necessary to any device. A page that has been marked up in TML can be ported to any device in the appropriate format. It imposes structure on content, encapsulating the details necessary to select and arrange content, allowing the structure to be easily extracted and presented to users independently of the end-user device. It is a powerful generic language (non-device specific) that produces a richly structured, loss-less representation of the original content. To summarise, TML:

- Extends XHTML.

- Provides a generic/non-device specific markup language.
- Imposes non-device specific structure on content.
- Aids easy transformation of content.

The optimised transcoder of Fig. 2 is modular, arising from an architecture having an object orientated Java module for each device format type. The modules inherit their behaviour from a base Mobility Module. This allows additional client-side capabilities to be added in a modular manner, thus minimising the lead time and development cost for accommodating new technologies.

Another major advantage of the transcoder is that it operates on-the-fly with a response time which is so short that transformation is transparent to the user.

Referring to Fig. 3, the transcoder is shown in more detail. The following are major operations.

A request is received.

A call chain is assembled for the request, the chain comprising transformation components chosen according to the type of device making the request. This avoids routing the request through unnecessary shared code.

The call chain components have access to user-related data without the need for them to directly interface with the storage devices. This isolation is provided by various "accessors", namely a request accessor, a session accessor, a service accessor, and a device accessor.

A transformation module pools call chains to minimise setup cost because it re-uses existing chains for subsequent similar chains.

The accessors are also pooled. In addition, an accessor is not instantiated unless it is absolutely required, "lazy-evaluated".

Each component in the call chain performs a very specific and minimal task, thus allowing excellent on-the-fly performance. Indeed the users perception is that the content was originally authored specifically for his or her device type and is thus transferred without transcoding.

In addition, the transcoder is capable of recognising when two or more users are requesting the same content on the same device type, and to cache the content for a rapid response. Cacheing is configured for the required granularity, such as page or content source granularity. For example, in the case where end-user requested content is aggregated into a page by combining static and dynamic content, the static elements of the page may be cached while particular dynamic content objects, requiring real-time compilation (e.g. latest stock values) can be aggregated 'on-the-fly' by the content transformation process.

The following aspect of the transcoder allows it to obtain sufficient efficiency to enable it to perform on-the-fly transformation transparently to the user.

- It minimises the amount of shared code which needs to be executed for each request.
- There is modular retrieval of user, service, device, group, session and request information so as to minimise the retrieval of unnecessary data.
- It caches data intelligently – so as to minimise the unnecessary retrieval of data when two or more users wish to access the same information.
- It intelligently transforms TML to device specific formats (e.g. WML, HTML, cHTML) using an optimised transforming method.

The following describes how each of these strategies are accomplished.

Minimise Shared Code

Traditional software design methodologies attempt to reduce the size of the development task through a process of functional decomposition – in other words, by identifying the set of all atomic functions required to complete a task and then architecting a cohesive solution which maximises the sharing of these functions throughout the code. This often results in code which approximates a railway points depot – i.e. numerous decision points exist where execution is gradually rerouted down one track or another. Figure 4 presents this concept graphically.

Thus, by applying traditional methods, a request from a users device to the transcoding engine would be routed through numerous decision points (and much shared code) to assess the type of device, the type of transcoding necessary, whether or not page-splitting will be required, whether images can be delivered to the device or not, etc.

Instead of this, the transcoder implements a dynamic 'call-chain' architecture. In other words, when a request is received by the transcoder, it identifies the type of device and immediately establishes the absolute minimal set of functions which will be required to service the request. Thus a single 'straight-line' execution is achieved, with no unnecessary decision points or corresponding evaluation overhead. By pre-loading the call-chains and implementing intelligent 'pooling', through the use of a call-chain factory, enormous efficiencies are achieved. Fig. 5 presents the call-chain which is dynamically instantiated for the WAP request mentioned earlier.

Modularise Data Retrieval using Accessors

Each request which is received by the transcoder requires varying degrees of information about the end-user, the users session, the device and the service that is being interrogated.

Traditional software design suggests that objects are created to contain the necessary information whenever a request is received. The inventors, however, recognise that creating these objects for every request is inefficient. Instead, data objects are created only when they are needed – additionally, they are pooled for reuse once the request has been serviced.

Data objects are accessed via lightweight 'Accessor' classes. Each time a request is received, a thin accessor is created to provide an interface to the data object. Data objects are only created when the requested information is not available from the data object pool. Considering that a finite number of device types exist and a finite number of services will be configured on a single server, there is sizeable scope for reuse of data objects.

It should also be noted that the accessors are created only when needed by a process of 'lazy evaluation' – for example, the transcoder will only create a User Accessor (an interface to the user-data object) when it is absolutely necessary (bear in mind that a high percentage of requests can be serviced without need to interrogate user-specific characteristics).

In the scenario of Fig. 6, the call chain described above requires only session, device and service accessors to be lazy-evaluated. User information is not required. The service accessor and the device accessor provide an interface to pooled data objects, thereby absolutely minimising the creation of data objects.

Intelligent Caching

The above description describes an approach to minimising the amount of processing required to pass a request to a service and transform the resultant data according to device capabilities.

In an environment where there exists a large community of users, a finite number of device types and a finite number of services, it is reasonable to assume that there will be a high percentage of requests to retrieve the same page of content on the same device by two or more users.

The transcoder provides a cache where transformed content can be stored (depending upon the level of sensitivity of the information, which can be configured at the service level). Additionally, the transcoder provides a number of mechanisms for recognising a request for information which already exists in a transformed state in the cache.

This enables the transcoding process to be further streamlined. Each request results in a cursory check upon the content cache. If the required information is contained in the cache, call-chain evaluation is arrested and no further accessors or data objects are created. The information is quickly retrieved from the cache and delivered to the user.

It is worth noting that there are many points along the call-chain which can be configured to use the cache. For instance, the transcoder also supports the aggregation of content from multiple sources (consider the example of a news service which draws information from various news information providers). In this case, content is retrieved from each of the services and is transcoded, first into TML and later into the device-specific dialect.

The efficiency of aggregating such information and transcoding it for the target device can be increased enormously by applying intelligent caching of the canonical data. Fig. 7 presents this graphically.

Intelligent Transformation

A standard industry approach to transforming one form of XML content to another is to utilise XSLTs (XML Style-sheet Transformation modules). In terms of processing, this involves:

1. Reading the source XML document.
2. Parsing this XML document using an XML parser and building a DOM tree from it.
3. Loading the XSLT
4. Applying the XSLT to the XML DOM to produce the transformed XML DOM. This is the most time consuming aspect of this type of transformation. The XSL processor wastes a huge amount of time searching through the XML DOM trying to match tags (which very often aren't even present) and then has to continually refer back to the transformation XSLT to try and interpret what sort of transformation steps need to be taken.
5. Serialising the transformed XML DOM for output.

Due to the complexities of the transformations (in particular the WML transformation) taking place and the limitations of XSLT, this approach is not efficient.

The transformation process of the invention below provides an approach to getting from 1 to 5, above more quickly and more efficiently. This process is termed "Intelligent DOM" in this specification.

Intelligent DOM

The Intelligent DOM transformation improves upon the standard industry approach of utilising an XSLT for transformation.

In any DOM model there exists the concept of a document, which is a tree like structure, made up of, amongst other things, elements, which are an object representation of markup tags. In the case of a standard DOM these elements are all of the same type even though the tags are all different, have different features and different degrees of relevance (especially in the context of a transformation). For example a JDOM (Java DOM) representation of a TML document is illustrated in Fig. 8.

As can be seen JDOM will create a tree like structure with every tag/element on the tree represented as an instance of Element. This is acceptable and a programmatic transformation, which would be faster than an XSLT transformation, could be carried out on the tree as it is. However, the transformation would have to be carried out in a flat, inefficient manner and the code would be difficult to maintain. To locate a particular node, a search would need to be carried out (probably starting at the root) in which the overhead of a string comparison would be incurred at every element.

The Intelligent DOM approach is different however. Instead of adding the same type of element object for every tag, an element object tailored specifically to handle that type of tag, is added to the tree. This is shown in Fig. 9.

Each of these special elements knows how to transform itself rather than depending on a "jack of all trades" to perform the transformation on all of them – in Object-

Oriented parlance, "Specialisation". Each of the special elements are specialised versions of the TMLElement class – i.e. extended from TMLElement.

With the Intelligent DOM the transformation can take place in two ways:

1. Transform the DOM as it is being built. As the elements are hit, while the DOM is being constructed, transform them, therefore going straight to a channel specific DOM.
2. Transform the DOM after it's built. Non-channel specific DOM – ideal for caching.

The invention is not limited to the embodiments described but may be varied in construction and detail.

Claims

1. A transcoder comprising:-

a request handler for receiving a request from a user device for content from a server, and for retrieving the requested content, and

a transformation means for dynamically transcoding the content to a format compatible with the device.

2. A transcoder as claimed in claim 1, wherein the transformation means comprises means for transcoding from a task markup language format describing the content inherent in a number of formats such as HTML, XHTML basic, WML, and plain text.

3. A transcoder as claimed in claims 1 or 2, wherein the request handler comprises means for generating a call chain of request-handling components chosen according to the type of user device.

4. A transcoder as claimed in any preceding claim, wherein the transformation means comprises means for pooling call chains for re-use.

5. A transcoder as claimed in any preceding claim, wherein the transformation means comprises means for using both user personalisation data and device data to perform transcoding.

6. A transcoder as claimed in claim 5, wherein the transcoder comprises a plurality of accessors, each comprising means for retrieving data in a manner which isolates the call chain components from the device data and personalisation data databases.

7. A transcoder as claimed in claim 6, wherein a thin accessor is instantiated as a lightweight façade to an object-orientated class.

8. A transcoder as claimed in claims 6 or 7, wherein an accessor is not instantiated unless it is required, "lazy-evaluated".

9. A transcoder as claimed in any of claims 6 to 8, wherein the transformation means comprises means for pooling instantiated accessors.

10. A transcoder as claimed in any preceding claim, wherein the transformation means comprises means for caching transformed content, and for preventing transformation from taking place if relevant content is in a cache.

11. A transcoder as claimed in claim 10, wherein the transformation means comprises means for checking a cache at a plurality of stages in a call chain.

12. A transcoder as claimed in any preceding claim, wherein the transformation means comprises means for building a Document Object Model (DOM) tree.

13. A transcoder as claimed in claim 11, wherein the tree is built with elements which have transformation intelligence.

14. A method of providing content to a user device, the method comprising the steps of:-

authoring content in a generic markup language; and

transforming the content on-the-fly in real time to a device format according to user preferences and device capabilities.

15. A method as claimed in claim 14, wherein the content is aggregated with content in a specific markup language before transformation.

16. A computer program product comprising software code for performing the steps of the method of claim 13 when executing on a digital computer.

ABSTRACT

"A content transformation system and method"

(Fig. 3)

Content is authored in a generic task markup language (TML). It is transformed on-the-fly to a format according to a mobile device capability and user preferences. In the transcoder, a call chain is established and there is pooling of call chain components. There is also cacheing of previously-retrieved content.

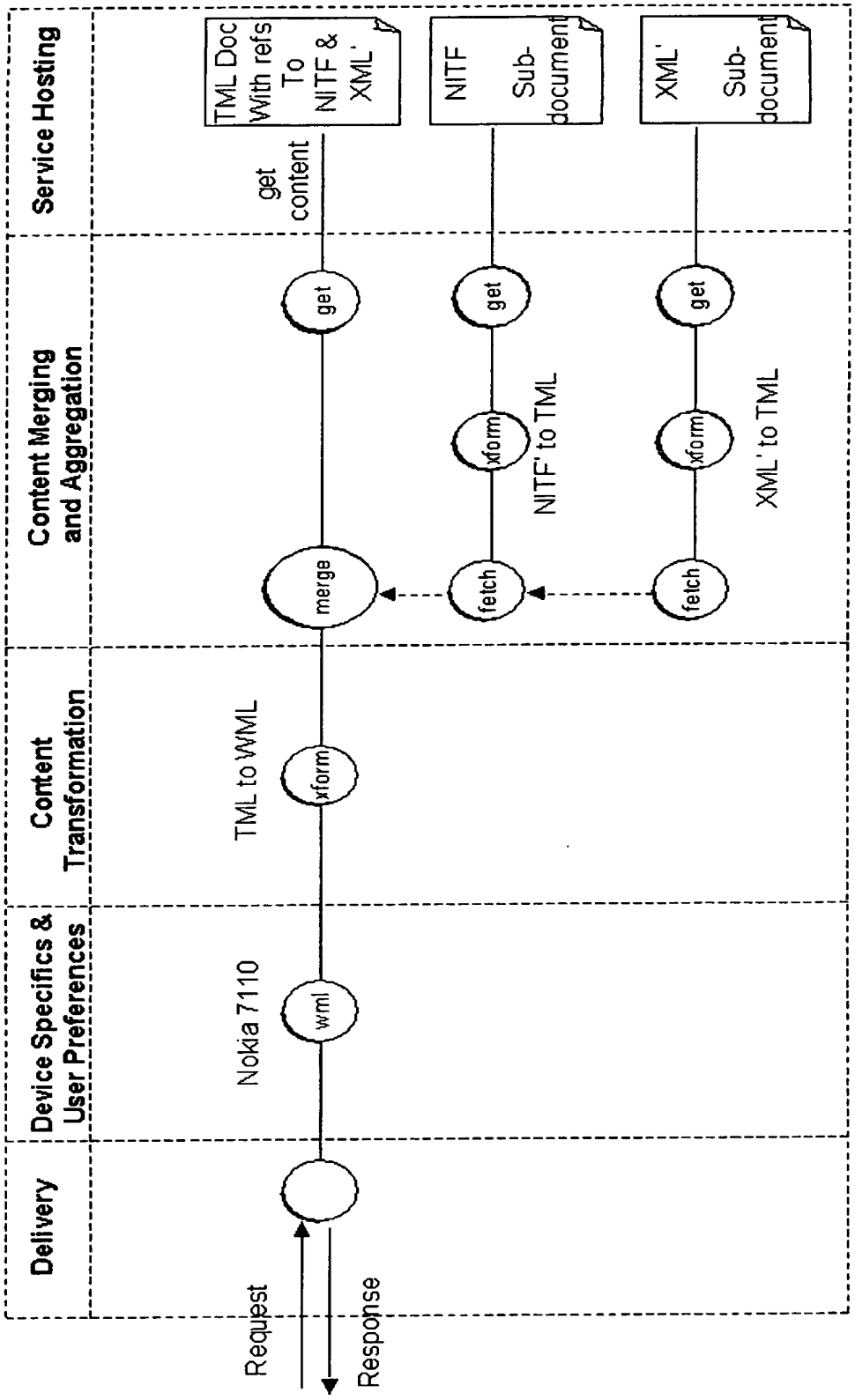
Service Hosting
Content Merging and Aggregation
Content Transformation
Device Specifics & User Preferences
Delivery
TML Doc With refs To NITF & XML'
NITF Sub-document
XML' Sub-document
get content
get
get
get
xform
xform
merge
fetch
fetch
NITF' to TML
XML' to TML
TML to WML
Nokia 7110
xform
wml
Request
Response

Fig. 1

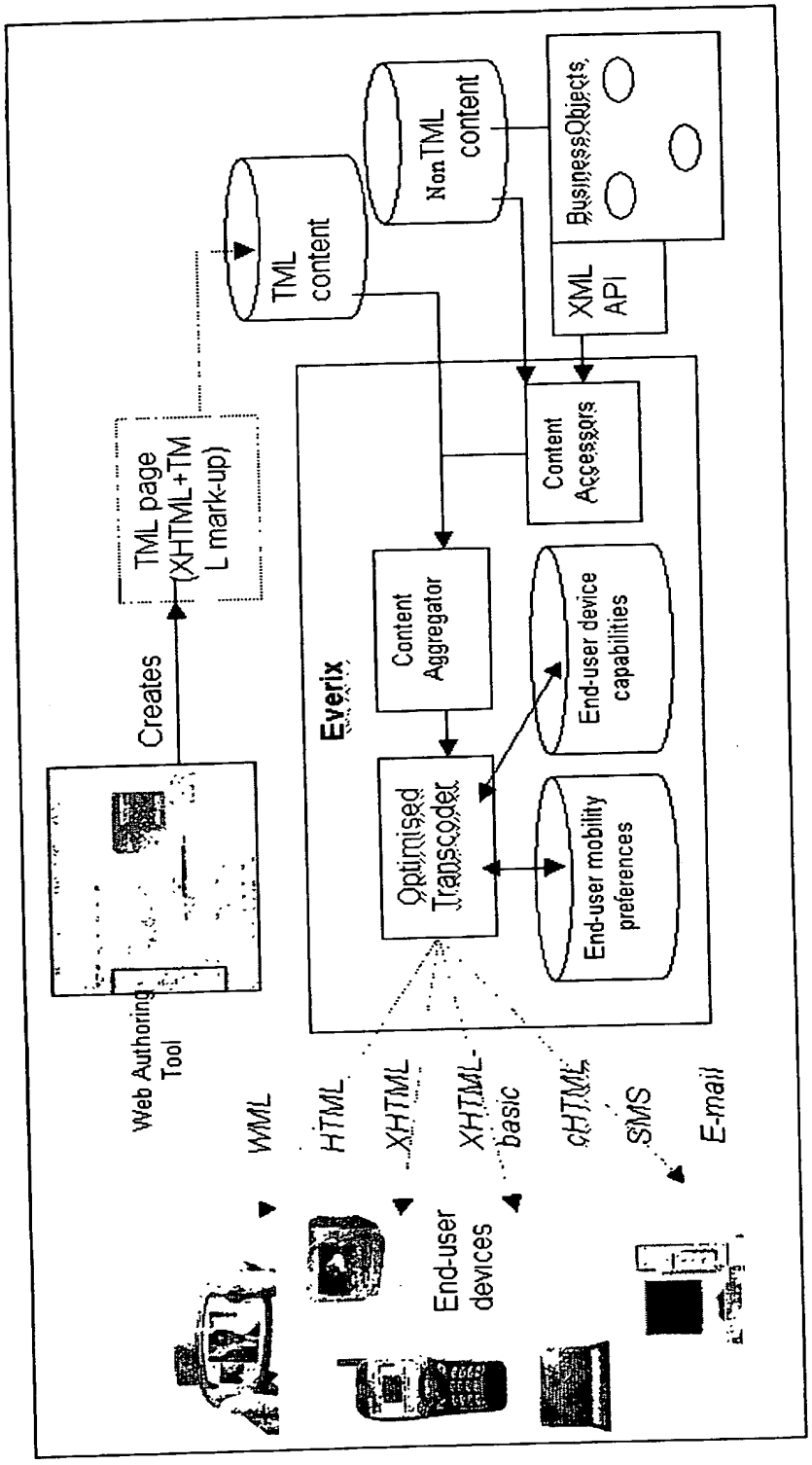
Web Authoring Tool
Creates
TML page (XHTML+TML mark-up)
Everix
Optimised Transcoder
Content Aggregator
Content Accessors
End-user device capabilities
End-user mobility preferences
TML content
Non TML content
XML API
BusinessObjects
WML
HTML
XHTML
XHTML-basic
cHTML
SMS
E-mail
End-user devices

Fig. 2

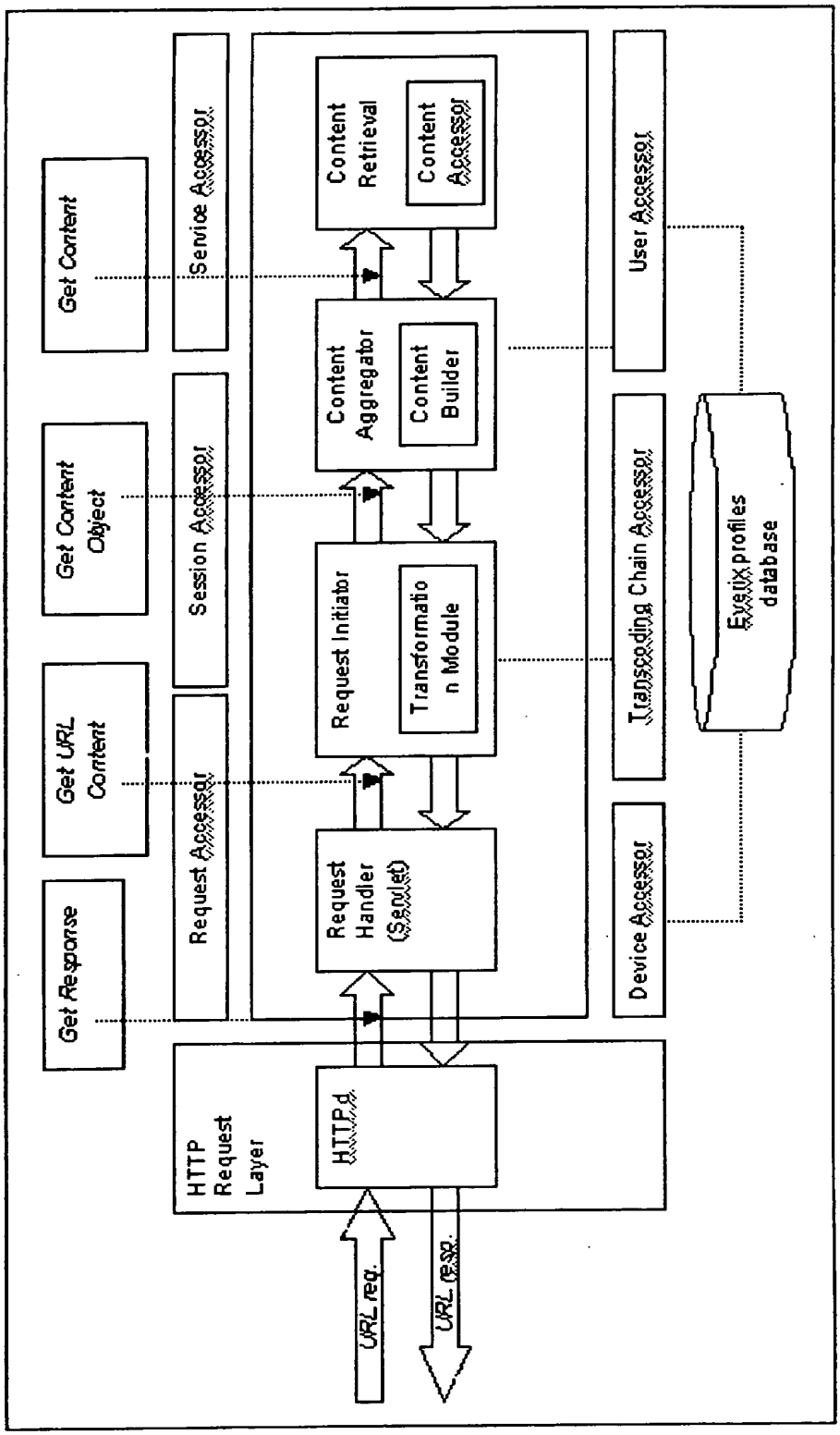
Get Content
Get Content Object
Get URL Content
Get Response
Service Accessor
Session Accessor
Request Accessor
Content Retrieval
Content Accessor
Content Aggregator
Content Builder
Request Initiator
Transformatio n Module
Request Handler (Servlet)
HTTP Request Layer
HTTPd
URL req.
URL resp.
User Accessor
Transcoding Chain Accessor
Device Accessor
Evolix profiles database

Fig. 3

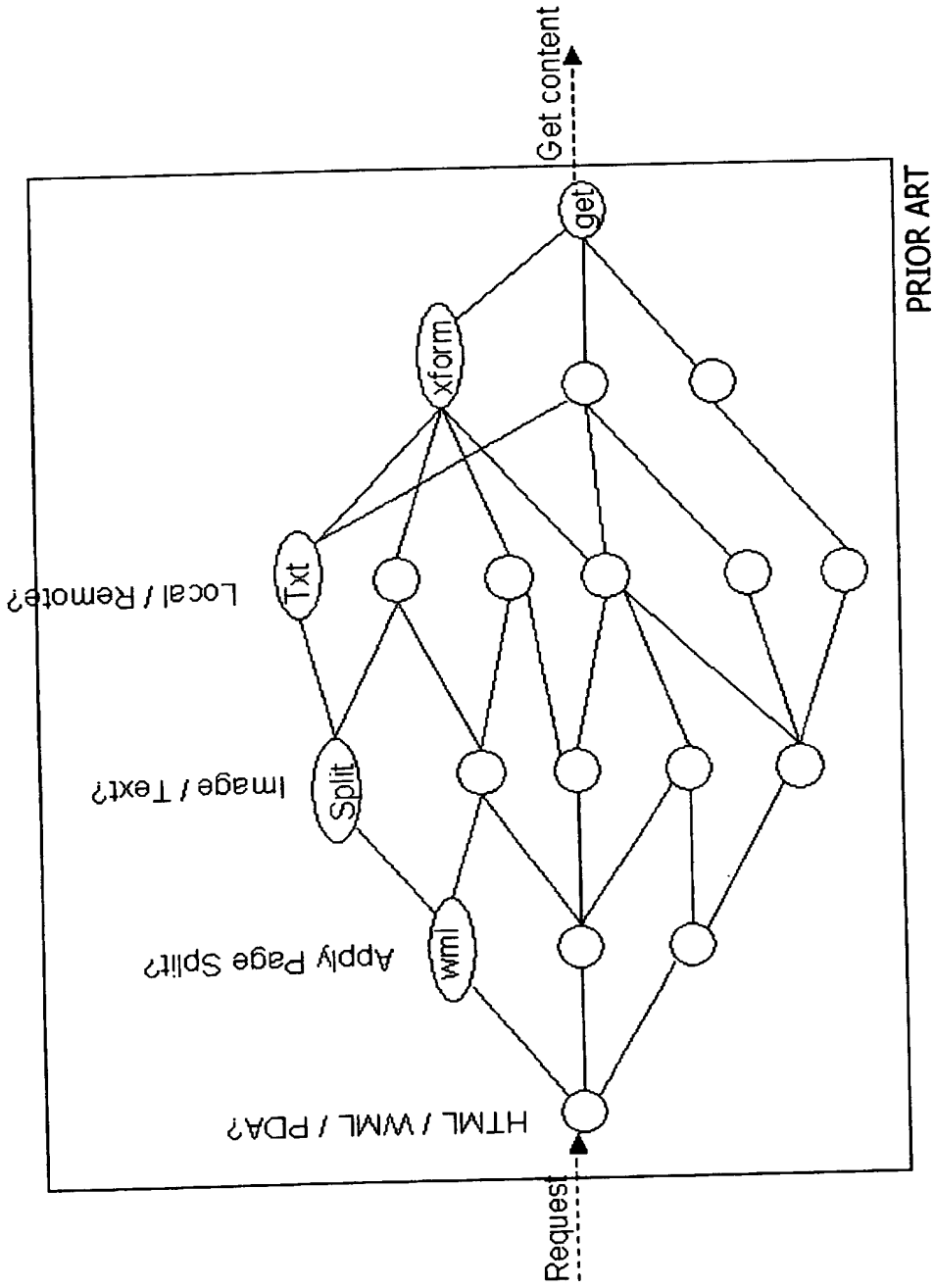
Get content
get
xform
Txt
Split
wml
Local / Remote?
Image / Text?
Apply Page Split?
HTML / WML / PDA?
Request
PRIOR ART

Fig. 4

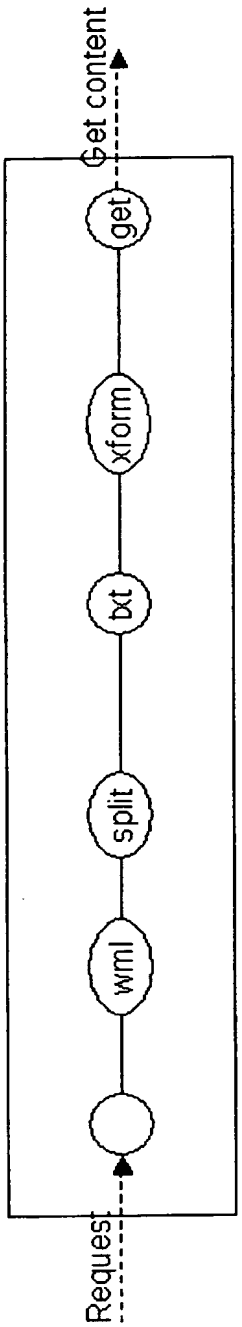
Get content
get
xform
txt
split
wml
Request

Fig. 5

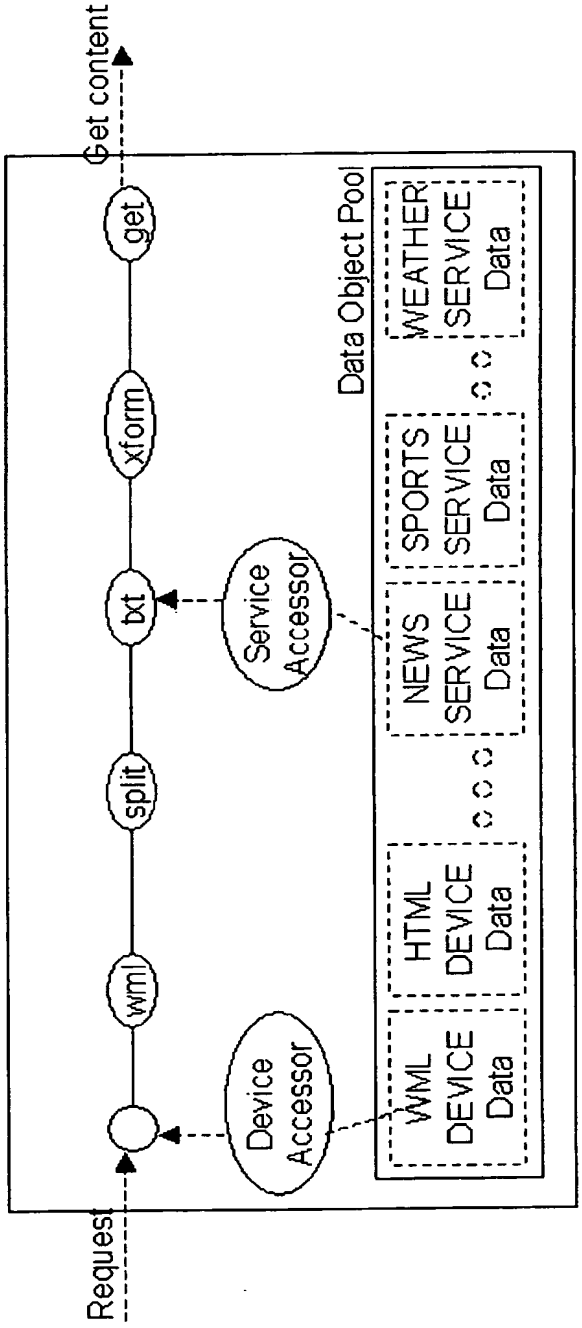
Get content
get
xform
txt
split
wml
Request
Service Accessor
Device Accessor
Data Object Pool
WML DEVICE Data
HTML DEVICE Data
NEWS SERVICE Data
SPORTS SERVICE Data
WEATHER SERVICE Data

Fig. 6

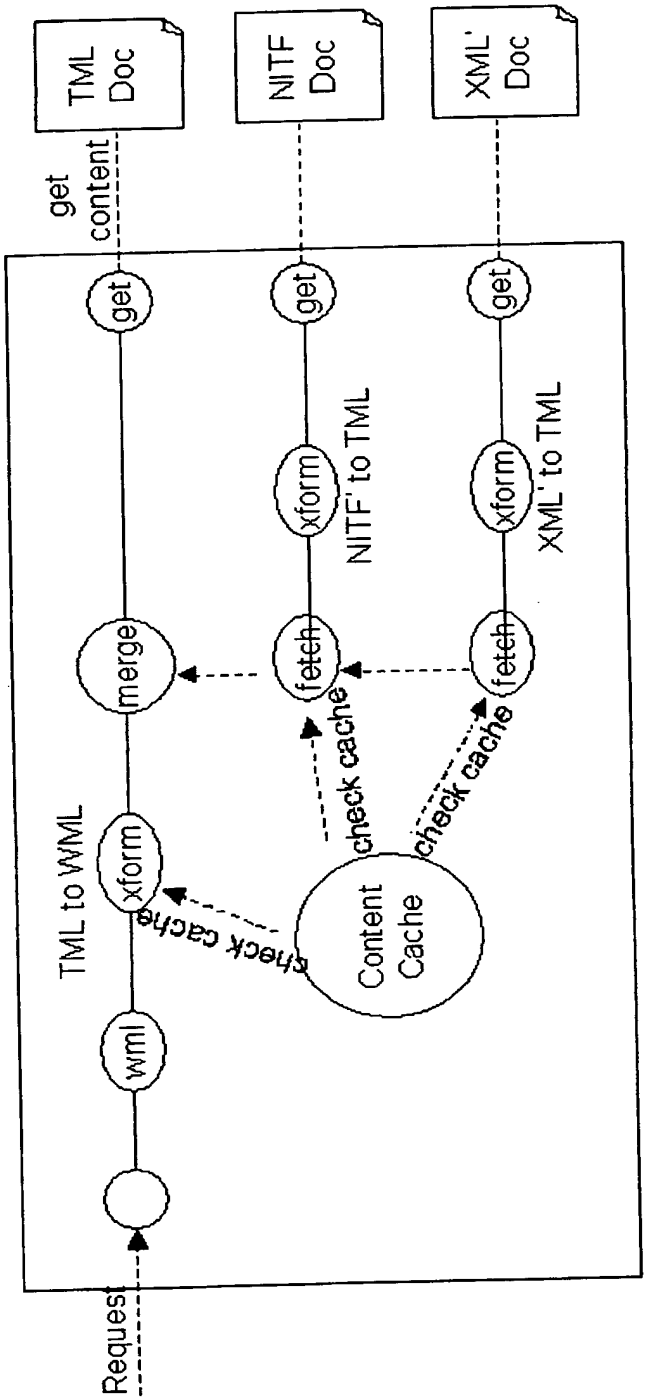
TML Doc
NITF Doc
XML' Doc
get content
get
get
get
xform
NITF' to TML
xform
XML' to TML
merge
fetch
fetch
check cache
check cache
TML to WML
xform
check cache
Content Cache
wml
Request

Fig. 7

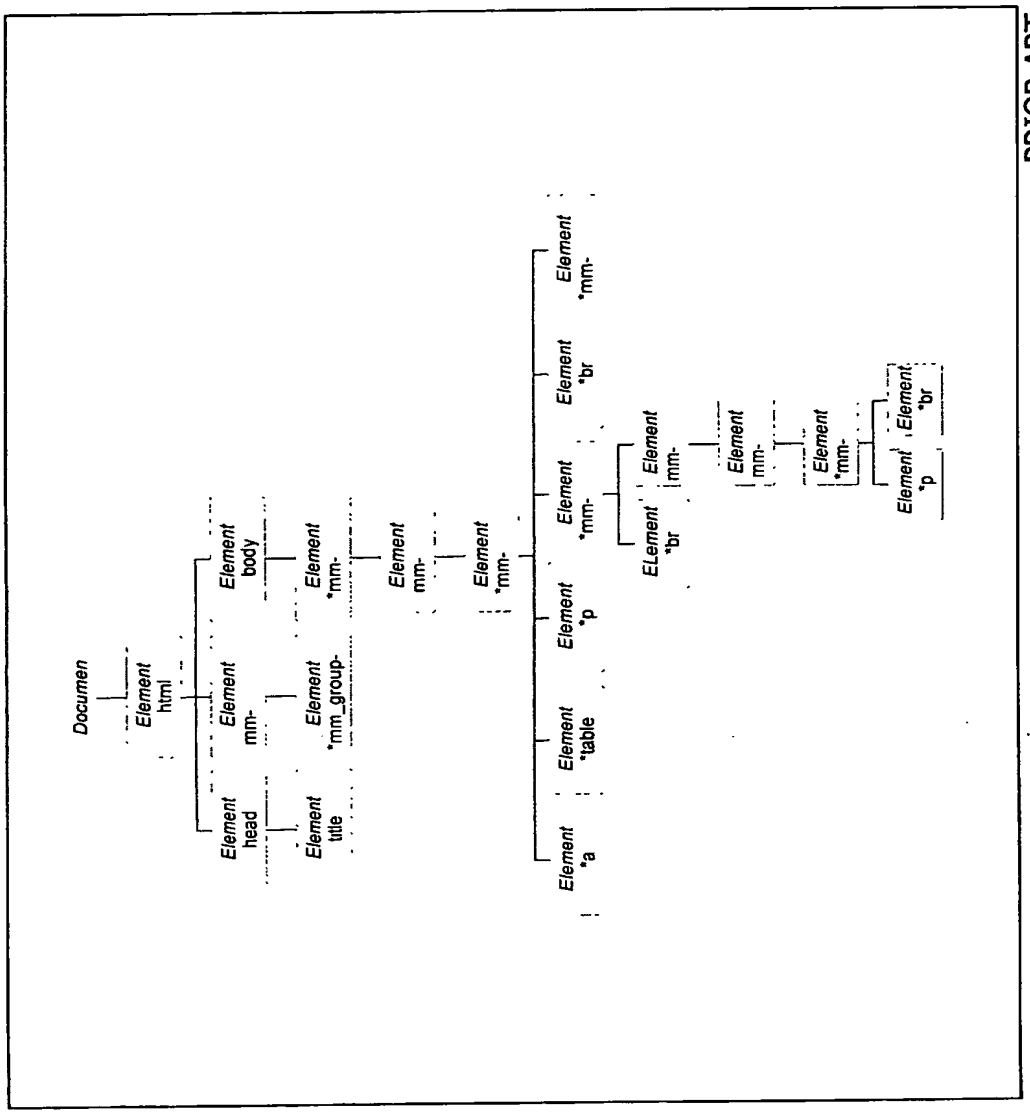
Documen
Element html
Element head
Element title
Element mm-
Element *mm_group-
Element body
Element *mm-
Element mm-
Element *mm-
Element *a
Element *table
Element *p
Element *mm-
ELement *br
Element mm-
Element *mm-
Element *br
Element mm-
Element mm-
Element *mm-
Element *p
Element *br
PRIOR ART

Fig. 8

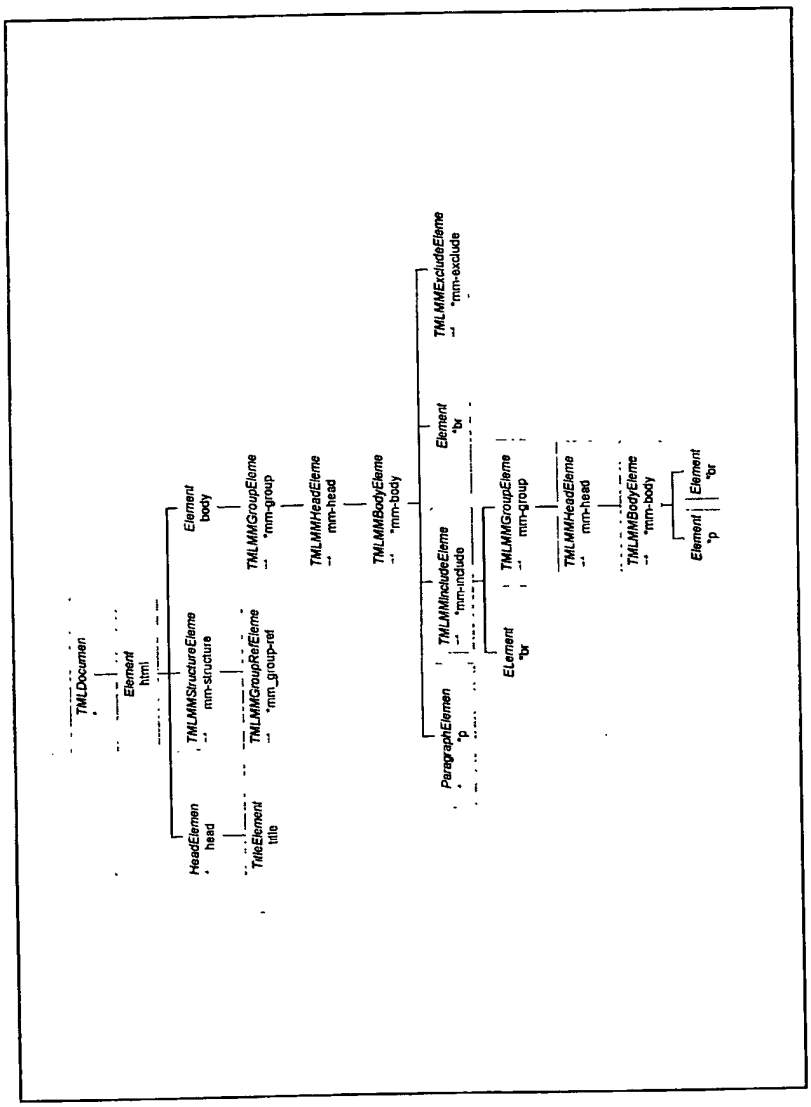
TMLDocumen
Element
html
HeadElemen
head
TMLMMStructureEleme
mm-structure
TitleElement
title
TMLMMGroupRefEleme
*mm_group-ref
Element
body
TMLMMGroupEleme
*mm-group
TMLMMHeadEleme
mm-head
TMLMMBodyEleme
*mm-body
ParagraphElemen
*p
TMLMMIncludeEleme
*mm-include
Element
*br
TMLMMExcludeEleme
*mm-exclude
Element
*br
TMLMMGroupEleme
mm-group
TMLMMHeadEleme
mm-head
TMLMMBodyEleme
*mm-body
Element
*p
Element
*br

Fig. 9